OBJECT FILE FORMATS

An INTEL® Software Standard

Printed in U.S.A./S310/0877/500 TP

98-183A

PREFACE

This manual defines the internal format of object files produced by Intel's resident or cross-product language translators and read by other Intel software products. The information in this manual is normally not needed in order to use Intel software but is provided for the person who needs to write a program to read these object files or to create files in the same format.

The character set of the American Standard Code for Information Interchange (ASCII) is defined in the following document:

American National Standard Institute, Code for Information Interchange, X3.4-1968

TABLE OF CONTENTS

ii PREFACE

1 CHAPTER 1: INTRODUCTION TO OBJECT FILE FORMATS

2 CHAPTER 2: HEXADECIMAL OBJECT FILE FORMAT FOR PAPER TAPE

4 CHAPTER 3: BNPF OBJECT FILE FORMAT FOR PAPER TAPE

CHAPTER 1: INTRODUCTION TO OBJECT FILE FORMATS

The formats defined in this standard are for object files produced by Intel's language translators. An object file contains a form of machine language instructions and/or data that permits loading into memory for execution or interpretation. In addition, it contains control information that may govern the loading process, such as load addresses and a starting address (the address at which execution is to begin).

An object file can be used to program a ROM or PROM. In this case it is loaded into RAM not for execution but simply as data for the program that transfers it to the ROM or PROM. The RAM address at which it is loaded is usually not the one at which the program will execute properly.

CHAPTER 2: HEXADECIMAL OBJECT FILE FORMAT FOR PAPER TAPE

Hexadecimal object file format is a way of representing a object file in ASCII. The ASCII character set is defined by the reference listed in the preface.

The hexadecimal representation of binary is coded in ASCII. For example, the 8-bit binary value 0011 1111 is 3F in hexadecimal. To code this in ASCII one 8-bit byte containing the ASCII code for 3 and one 8-bit byte containing the ASCII code for F are required. This representation (ASCII hexadecimal) requires twice as many bytes as the binary.

A hexadecimal object file can contain either 8-bit or 4-bit data but not both. Two ASCII hexadecimal digits are used to represent both 8-bit and 4-bit data. In the case of 4-bit data, only one of the digits is meaningful. Whether it is the high-order or the low-order digit must be known by the program reading the file and must be consistent throughout the file.

Since ASCII characters need only 7-bits for their representation, the highest-order bit of each 8-bit byte can be used as a parity bit by a program that generates the hexadecimal format object file. However, when such a file is loaded by an Intel product, the highest order bit is masked as the ASCII is converted to binary. Also, Intel software does not generate parity bits when creating object files.

The format described below is for paper tape and does not define the format for other media, which may use record separators such as the ASCII code for carriage return. On paper tape, one ASCII character requires one frame. The record format is described here according to the fields in the record.

The object code file contains two parts: the object code preceded by a symbol table. The symbol table is a series of records terminated by a dollar sign. Each record contains three fields separated by one or more ASCII spaces: a number field, a label field, and an address field. All fields must be present although they may contain dummy values (such as zero) if not applicable. The number field contains a combination of ASCII digits from 0-9. The label field contains ASCII characters representing a symbol from the source program or a symbol generated by the language translator. The address field contains a value that must begin with a decimal digit (0-9) and whose last character signifies the base as follows:

H	Hexadecimal
O or Q	Octal
B	Binary
D	Decimal

If the value does not end with one of these characters, the base is decimal. The symbol table is terminated by a record whose first nonblank character is a dollar sign.

Following the symbol table is the object code, which may be preceded and followed by a row of asterisks (so the object code can be visually located on paper tape). Each record in the object code may be preceded by ASCII spaces. The fields in each record are described in the following paragraphs.

RECORD MARK FIELD: Frame 0

The ASCII code for a colon (:) is used to signal the start of a record.

RECORD LENGTH FIELD: Frames 1 and 2

The number of data bytes in the record is represented by two ASCII hexadecimal digits in this field. The high-order digit is in frame 1. The maximum number of data bytes in a record is 255 (FF in hexadecimal). An end-of-file record contains two ASCII zeros in this field.

LOAD ADDRESS FIELD: Frames 3 - 6

The four ASCII hexadecimal digits in frames 3-6 give the address at which the data is loaded. The high-order digit is in frame 3, the low-order digit in frame 6. The first data byte is stored in the location indicated by the load address; successive bytes are stored in successive memory locations. This field in an end-of-file record contains zeros or the starting address of the program.

RECORD TYPE FIELD: Frames 7 and 8

The two ASCII hexadecimal digits in this field specify the record type. The high-order digit is in frame 7. All data records are type 0; end-of-file records are type 1. Other possible values for this field are reserved for future expansion.

DATA FIELD: Frames 9 to 9+2*(record length)-1

A data byte is represented by two frames containing the ASCII characters 0-9 or A-F, which represent a hexadecimal value between 0 and FF (0 and 255 decimal). The high-order digit is in the first frame of each pair. If the data is 4-bit, then either the high or low-order digit represents the data and the other digit of the pair may be any ASCII hexadecimal digit. There are no data bytes in an end-of-file record.

CHECKSUM FIELD: Frames 9+2*(record length) to 9+2*(record length)+1

The checksum field contains the ASCII hexadecimal representation of the twos complement of the 8-bit sum of the 8-bit bytes that result from converting each pair of ASCII

hexadecimal digits to one byte of binary, from the record length field to and including the last byte of the data field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the record length field to and including the checksum field, is zero.

The following example shows a hexadecimal object code file with each record on a separate line.

```
0 BLOCK01 0
0 ACTUA 0318CH
0 AFT   0317EH
0 BEGIN 03100H
0 BUFFE 03196H
0 CAFT  0317AH
0 CBLK  0317AH
0 CLOSE 00001H
0 DONE  03150H
0 EBLK  03192H
0 ERR   03160H
0 ERROR 0000CH
0 EXIT  00009H
0 ISIS  00040H
0 LOOP  03127H
0 OBLK  03170H
0 OPEN  00000H
0 RBLK  0317EH
0 READ  00003H
0 STACK 03216H
0 STATU 03192H
0 WBLK  03188H
0 WRITE 00004H
0 XBLK  03190H
 $
:10310000311A320E03117E31CD40003A9231B7C2EE
:1031100060310E00117031CD40003A9231B7C2607B
:10312000312A7E31227A310E03117E31CD40003AB0
:103130009231B7C260312A8C317CB5CA50310E044D
:10314000118831CD40003A9231B7C26031C3273186
:103150000E01117A31CD40000E09119031CD4000A1
:103160000E0C119231CD40000E09119031CD40006E
:0A3170007E3196310100000092311B
:10317C009231010096318000 8C31923100009631F1
:04318E0092319231B7
:02319400923176
:00310001CE
```

CHAPTER 3: BNPF OBJECT FILE FORMAT FOR PAPER TAPE

The BNPF object file format contains two parts: the object code preceded by a symbol table. Only the object code is in BNPF. The format of the symbol table is identical to that for hexadecimal object files described in Chapter 2.

The BNPF object code uses the ASCII characters N and P to represent the actual binary digits 0 and 1 respectively. An ASCII "B" indicates the beginning of a byte, an ASCII "F" indicates the end of the byte. All characters following the F are ignored until another B is encountered. This allows comments that do not contain the character B to appear between bytes of data in BNPF format.

Ten bytes (frames on paper tape) are required to represent one byte of data. For example, the byte containing the value 0011 1111 (3F in hexadecimal) would be represented as follows in BNPF:

```
BNNPPPPPPF
```

Exactly 8 characters must occur between the B and its following F. No character other than N or P can be used between the B and the F.

A BNPF object file may contain either 8-bit or 4-bit data but not both. In the case of 4-bit data, only the high-order or low-order 4 bits are meaningful. Which bits are meaningful must be known to the program reading the file and must be consistent throughout the file.